ÉCOLE SUPÉRIEURE DE TECHNOLOGIE
Fkih Ben Salah

TP 2 – Structures en C

IDS – Structures de Données en C

Compte Rendu

Ajournée: oufella mohamed

Introduction

Ce TP a pour objectif de maîtriser l'utilisation des structures (struct) en langage C. Les structures permettent de regrouper plusieurs variables de types différents sous un même nom. Dans ce TP, nous travaillerons avec une structure Etudiant et nous verrons progressivement comment la manipuler : déclaration, saisie, affichage, tri, recherche, allocation dynamique et listes chaînées.

Exercice 1 – Définir une structure Etudiant et afficher ses informations

Objectif

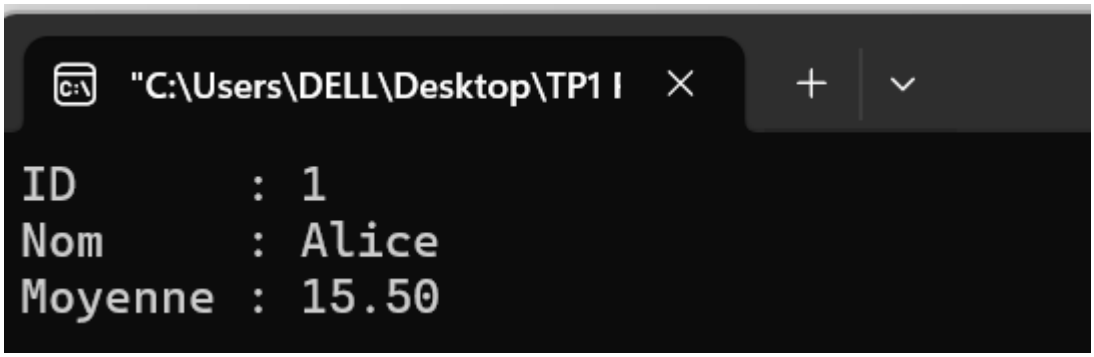
Déclarer une structure Etudiant contenant les champs id (entier), nom (chaîne de caractères) et moyenne (flottant), initialiser une variable de ce type et afficher ses informations.

Code source

*main.c

```c
#include <stdio.h>

struct Etudiant {
    int id;
    char nom[50];
    float moyenne;
};

int main() {
    struct Etudiant e1 = {1, "Alice", 15.5};
    printf("ID     : %d\n", e1.id);
    printf("Nom    : %s\n", e1.nom);
    printf("Moyenne: %.2f\n", e1.moyenne);
    return 0;
}
```

Résultat attendu



Exercice 2 – Saisir les informations d'un étudiant et les afficher

Objectif

Utiliser scanf pour saisir les informations d'un étudiant au clavier, puis les afficher.

Code source

*main.c

```c
#include <stdio.h>

struct Etudiant {
    int id;
    char nom[50];
    float moyenne;
};

int main() {
    struct Etudiant e;
    printf("Entrez l'ID     : "); scanf("%d", &e.id);
    printf("Entrez le nom   : "); scanf("%s", e.nom);
    printf("Entrez la moyenne: "); scanf("%f", &e.moyenne);
    printf("\n--- Informations ---\n");
    printf("ID     : %d\n", e.id);
    printf("Nom    : %s\n", e.nom);
    printf("Moyenne: %.2f\n", e.moyenne);
    return 0;
}
```

Explication

On utilise scanf avec le bon format pour chaque type : %d pour int, %s pour char[], et %f pour float. Notez que pour les chaînes, on n'utilise pas &.

Exercice 3 – Tableau de 3 étudiants

Objectif

Créer un tableau de structures de 3 étudiants, les initialiser et afficher leurs données.

Code source

```c
#include <stdio.h>
struct Etudiant {
    int id;
    char nom[50];
    float moyenne;
};

int main() {
    struct Etudiant tab[3] = {
        {1, "Alice", 15.5},
        {2, "Bob",   12.0},
        {3, "Carol", 18.3}
    };
    for (int i = 0; i < 3; i++) {
        printf("ID: %d | Nom: %s | Moyenne: %.2f\n",
               tab[i].id, tab[i].nom, tab[i].moyenne);
    }
    return 0;
}
```

Exercice 4 – Trouver l'étudiant ayant la meilleure moyenne

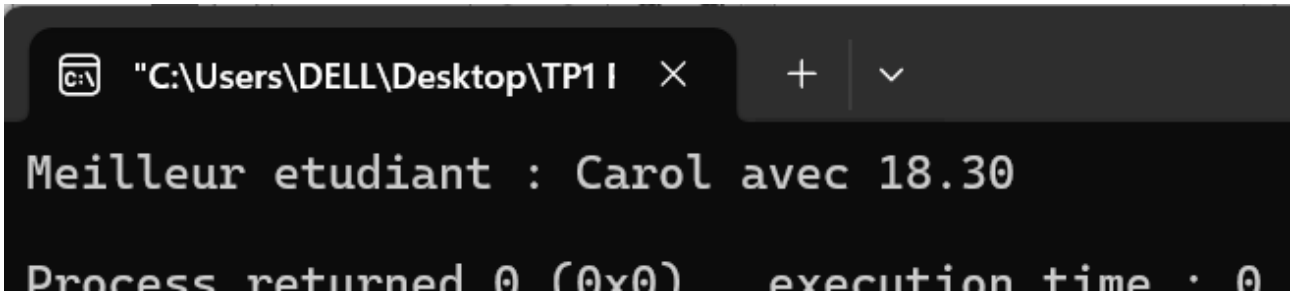
Objectif

Parcourir le tableau et identifier l'étudiant dont la moyenne est la plus élevée.

Code source

```c
#include <stdio.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

int main() {
    struct Etudiant tab[3] = {
        {1, "Alice", 15.5}, {2, "Bob", 12.0}, {3, "Carol", 18.3}
    };
    int idxMax = 0;
    for (int i = 1; i < 3; i++) {
        if (tab[i].moyenne > tab[idxMax].moyenne)
            idxMax = i;
    }
    printf("Meilleur etudiant : %s avec %.2f\n",
           tab[idxMax].nom, tab[idxMax].moyenne);
    return 0;
}
```

Résultat attendu



Exercice 5 – Fonction d'affichage d'un étudiant

Objectif

Définir une fonction afficherEtudiant qui prend une structure Etudiant en paramètre et l'affiche.

Code source

Start here | *main.c

```c
#include <stdio.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

void afficherEtudiant(struct Etudiant e) {
    printf("ID: %d | Nom: %s | Moyenne: %.2f\n",
           e.id, e.nom, e.moyenne);
}

int main() {
    struct Etudiant e = {1, "Alice", 15.5};
    afficherEtudiant(e);
    return 0;
}
```

Explication

La structure est passée par valeur à la fonction. Une copie locale est créée dans la fonction. C'est suffisant pour un simple affichage.

Exercice 6 – Modifier la moyenne via un pointeur

Objectif

Créer une fonction qui modifie la moyenne d'un étudiant en utilisant un pointeur sur structure.

Code source

Start here | main.c

```c
#include <stdio.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

void modifierMoyenne(struct Etudiant *e, float nouvelleMoyenne) {
    e->moyenne = nouvelleMoyenne;
}

int main() {
    struct Etudiant e = {1, "Alice", 15.5};
    printf("Avant : %.2f\n", e.moyenne);
    modifierMoyenne(&e, 18.0);
    printf("Apres : %.2f\n", e.moyenne);
    return 0;
}
```

Explication

L'opérateur -> permet d'accéder aux membres d'une structure via un pointeur. C'est équivalent à (*e).moyenne. Sans pointeur, la modification ne serait pas répercutée dans la variable d'origine.

Exercice 7 – Structure Date intégrée dans Etudiant

Objectif

Créer une structure Date (jour, mois, annee) et l'utiliser comme champ dans la structure Etudiant pour représenter la date de naissance.

Code source

t here main.c

```c
#include <stdio.h>

struct Date {
    int jour;
    int mois;
    int annee;
};

struct Etudiant {
    int id;
    char nom[50];
    float moyenne;
    struct Date dateNaissance;
};

int main() {
    struct Etudiant e = {1, "Alice", 15.5, {12, 3, 2002}};
    printf("Nom  : %s\n", e.nom);
    printf("Date : %02d/%02d/%d\n",
           e.dateNaissance.jour,
           e.dateNaissance.mois,
           e.dateNaissance.annee);
    return 0;
}
```

Résultat attendu

```
"C:\Users\DELL\Desktop\TP1 I
Nom  : Alice
Date : 12/03/2002

Process returned 0 (0x0)   execution time : 0.088 s
Press any key to continue.
```

Exercice 8 – Tableau dynamique de structures

Objectif

Allouer dynamiquement un tableau de n structures Etudiant avec malloc, saisir les données et les afficher, puis libérer la mémoire.

Code source

ere main.c

```c
#include <stdio.h>
#include <stdlib.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

int main() {
    int n;
    printf("Nombre d'etudiants : "); scanf("%d", &n);
    struct Etudiant *tab = malloc(n * sizeof(struct Etudiant));
    if (tab == NULL) { printf("Erreur allocation\n"); return 1; }
    for (int i = 0; i < n; i++) {
        printf("Etudiant %d :\n", i+1);
        printf("  ID      : "); scanf("%d", &tab[i].id);
        printf("  Nom     : "); scanf("%s", tab[i].nom);
        printf("  Moyenne: "); scanf("%f", &tab[i].moyenne);
    }
    printf("\n--- Liste ---\n");
    for (int i = 0; i < n; i++)
        printf("%d | %s | %.2f\n", tab[i].id, tab[i].nom, tab[i].moyenne);
    free(tab);
    return 0;
}
```

Explication

malloc(n * sizeof(struct Etudiant)) alloue n blocs contigus en mémoire. Il est indispensable de vérifier si l'allocation a réussi (pointeur non NULL) et de libérer la mémoire avec free() à la fin.

Exercice 9 – Trier les étudiants par moyenne

Objectif

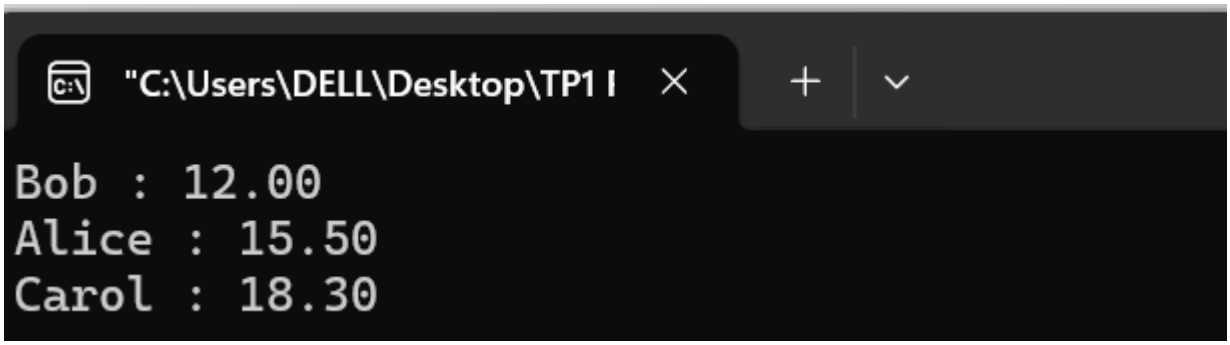
Trier un tableau de 3 étudiants par ordre croissant de moyenne en utilisant le tri à bulles.

Code source

here × main.c ×

```c
#include <stdio.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

void trierParMoyenne(struct Etudiant tab[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-1-i; j++) {
            if (tab[j].moyenne > tab[j+1].moyenne) {
                struct Etudiant temp = tab[j];
                tab[j] = tab[j+1];
                tab[j+1] = temp;
            }
        }
    }
}

int main() {
    struct Etudiant tab[3] = {
        {1,"Alice",15.5},{2,"Bob",12.0},{3,"Carol",18.3}
    };
    trierParMoyenne(tab, 3);
    for (int i = 0; i < 3; i++)
        printf("%s : %.2f\n", tab[i].nom, tab[i].moyenne);
    return 0;
}
```

Résultat attendu



Exercice 10 – Rechercher un étudiant par son ID

Objectif

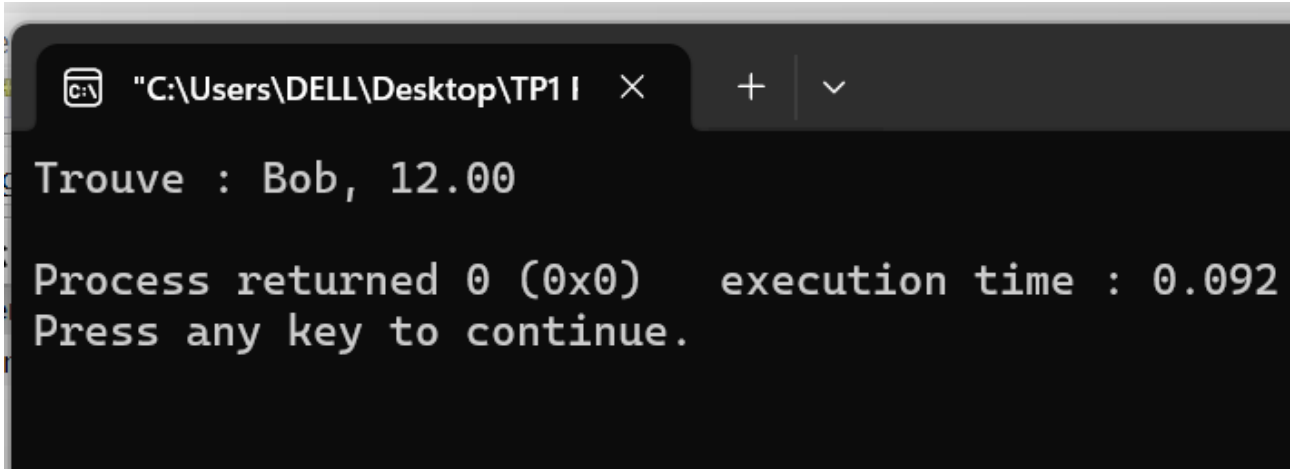
Écrire une fonction qui cherche un étudiant dans un tableau à partir de son identifiant, et retourne un pointeur vers cet étudiant (ou NULL si non trouvé).

Code source

main.c

```c
#include <stdio.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

struct Etudiant* rechercherParID(struct Etudiant tab[], int n, int id) {
    for (int i = 0; i < n; i++)
        if (tab[i].id == id) return &tab[i];
    return NULL;
}

int main() {
    struct Etudiant tab[3] = {
        {1,"Alice",15.5},{2,"Bob",12.0},{3,"Carol",18.3}
    };
    struct Etudiant *p = rechercherParID(tab, 3, 2);
    if (p != NULL)
        printf("Trouve : %s, %.2f\n", p->nom, p->moyenne);
    else
        printf("Etudiant non trouve.\n");
    return 0;
}
```

Résultat attendu



Exercice 11 – Créer un nœud d'une liste chaînée

Objectif

Définir une structure Noeud contenant un étudiant et un pointeur vers le nœud suivant, puis créer et afficher un seul nœud.

Code source

*main.c

```c
#include <stdio.h>
#include <stdlib.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

struct Noeud {
    struct Etudiant data;
    struct Noeud *suivant;
};

int main() {
    struct Noeud *n = malloc(sizeof(struct Noeud));
    n->data.id = 1;
    n->data.moyenne = 15.5;
    sprintf(n->data.nom, "Alice");
    n->suivant = NULL;
    printf("Noeud : %s, %.2f\n", n->data.nom, n->data.moyenne);
    free(n);
    return 0;
}
```

Explication

Le champ suivant de type struct Noeud* est le lien qui permet de chaîner les nœuds entre eux. Le dernier nœud de la liste pointe toujours vers NULL.

Exercice 12 – Ajouter un élément au début d'une liste chaînée

Objectif

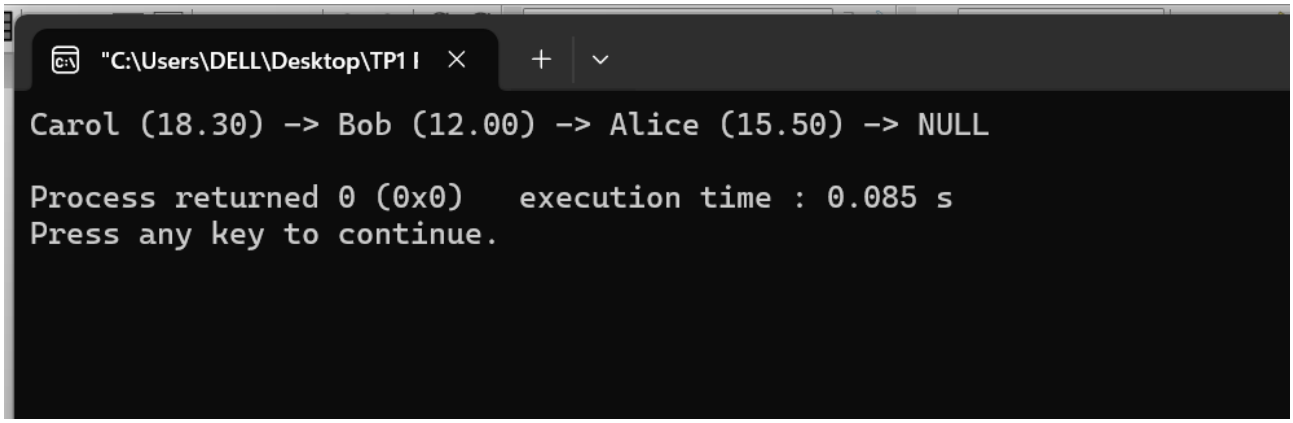
Écrire une fonction ajouterDebut qui insère un nouveau nœud au début d'une liste chaînée et retourne la nouvelle tête.

Code source

Start here | *main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Etudiant { int id; char nom[50]; float moyenne; };

struct Noeud {
    struct Etudiant data;
    struct Noeud *suivant;
};

struct Noeud* ajouterDebut(struct Noeud *tete, struct Etudiant e) {
    struct Noeud *nouveau = malloc(sizeof(struct Noeud));
    nouveau->data = e;
    nouveau->suivant = tete;
    return nouveau;
}

void afficherListe(struct Noeud *tete) {
    while (tete != NULL) {
        printf("%s (%.2f) -> ", tete->data.nom, tete->data.moyenne);
        tete = tete->suivant;
    }
    printf("NULL\n");
}

int main() {
    struct Noeud *liste = NULL;
    struct Etudiant e1 = {1, "Alice", 15.5};
    struct Etudiant e2 = {2, "Bob",   12.0};
    struct Etudiant e3 = {3, "Carol", 18.3};
    liste = ajouterDebut(liste, e1);
    liste = ajouterDebut(liste, e2);
    liste = ajouterDebut(liste, e3);
    afficherListe(liste);
    return 0;
}
```

Résultat attendu


"C:\Users\DELL\Desktop\TP1 I
Carol (18.30) -> Bob (12.00) -> Alice (15.50) -> NULL
Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.

Explication

L'insertion en début de liste est en O(1) : on crée un nouveau nœud, on fait pointer son champ suivant vers l'ancienne tête, puis on retourne ce nouveau nœud comme nouvelle tête.

Conclusion

Ce TP a permis d'explorer de façon progressive les structures en C :

- Déclaration et initialisation d'une struct
- Saisie et affichage avec scanf/printf
- Tableaux de structures et recherche
- Passage par pointeur pour modifier des données
- Imbrication de structures (Date dans Etudiant)
- Allocation dynamique avec malloc/free
- Tri par bulles sur des structures
- Listes chaînées : nœud et insertion en tête

La maîtrise des structures est fondamentale pour la mise en œuvre de toutes les structures de données avancées (listes, piles, files, arbres) abordées dans ce cours.